# 设计链表

## 题目

你可以选择使用单链表或者双链表，设计并实现自己的链表。

单链表中的节点应该具备两个属性：val 和 next 。val 是当前节点的值，next 是指向下一个节点的指针/引用。

如果是双向链表，则还需要属性 prev 以指示链表中的上一个节点。假设链表中的所有节点下标从 **0** 开始。

实现 MyLinkedList 类：

* MyLinkedList() 初始化 MyLinkedList 对象。
* int get(int index) 获取链表中下标为 index 的节点的值。如果下标无效，则返回 -1 。
* void addAtHead(int val) 将一个值为 val 的节点插入到链表中第一个元素之前。在插入完成后，新节点会成为链表的第一个节点。
* void addAtTail(int val) 将一个值为 val 的节点追加到链表中作为链表的最后一个元素。
* void addAtIndex(int index, int val) 将一个值为 val 的节点插入到链表中下标为 index 的节点之前。如果 index 等于链表的长度，那么该节点会被追加到链表的末尾。如果 index 比长度更大，该节点将 **不会插入** 到链表中。
* void deleteAtIndex(int index) 如果下标有效，则删除链表中下标为 index 的节点。

**示例：**

输入  
["MyLinkedList", "addAtHead", "addAtTail", "addAtIndex", "get", "deleteAtIndex", "get"]  
[[], [1], [3], [1, 2], [1], [1], [1]]  
输出  
[null, null, null, null, 2, null, 3]

解释  
MyLinkedList myLinkedList = new MyLinkedList();  
myLinkedList.addAtHead(1);  
myLinkedList.addAtTail(3);  
myLinkedList.addAtIndex(1, 2); // 链表变为 1->2->3  
myLinkedList.get(1); // 返回 2  
myLinkedList.deleteAtIndex(1); // 现在，链表变为 1->3  
myLinkedList.get(1); // 返回 3

**提示：**

* 0 <= index, val <= 1000
* 请不要使用内置的 LinkedList 库。
* 调用 get、addAtHead、addAtTail、addAtIndex 和 deleteAtIndex 的次数不超过 2000

## 题解

单链表：

class MyLinkedList {  
public:  
 MyLinkedList() {  
 this->size=0;  
 this->head=new ListNode(0);  
 }  
   
 int get(int index) {  
 //合法化判断  
 if(index<0||index>=size) return -1;  
 ListNode \*p=head;  
 for(int i=0;i<index;i++)  
 {  
 p=p->next;  
 }  
 return p->next->val;  
 }  
   
 void addAtHead(int val) {  
 addAtIndex(0,val);  
 }  
   
 void addAtTail(int val) {  
 addAtIndex(size,val);  
 }  
   
 void addAtIndex(int index, int val) {  
 if(index<-1||index>size) return;  
 ListNode \*p=head;  
   
 for(int i=0;i<index;i++)  
 {  
 p=p->next;  
 }  
 ListNode \*q=new ListNode(val,p->next);  
 p->next=q;  
 size++;  
 }  
   
 void deleteAtIndex(int index) {  
 if(index<0||index>=size) return;  
 ListNode \*p=head;  
   
 for(int i=0;i<index;i++)  
 {  
 p=p->next;  
 }  
 //这里注意一下  
 ListNode \*q=p->next;  
 p->next=p->next->next;  
 delete q;  
 size--;  
 }  
 private:  
 int size;  
 ListNode \*head;  
};  
/\*\*  
 \* Your MyLinkedList object will be instantiated and called as such:  
 \* MyLinkedList\* obj = new MyLinkedList();  
 \* int param\_1 = obj->get(index);  
 \* obj->addAtHead(val);  
 \* obj->addAtTail(val);  
 \* obj->addAtIndex(index,val);  
 \* obj->deleteAtIndex(index);  
 \*/

双链表：

typedef struct dLinkedListNode{  
 int val;  
 dLinkedListNode \*pre;  
 dLinkedListNode \*aft;  
 dLinkedListNode(int \_val) : val(\_val), pre(nullptr), aft(nullptr) {}  
}dll;  
class MyLinkedList {  
public:  
 MyLinkedList() {  
 this->size=0;  
 this->head=new dll(0);  
 this->tail=new dll(0);  
 head->aft=tail;  
 tail->pre=head;  
 }  
   
 int get(int index) {  
 //合法化判断  
 if(index<0||index>=size) return -1;  
 dll \*p;  
 if(index+1<size-index)//也就是在head离他更近一点  
 {  
 p=head;  
 for(int i=0;i<=index;i++)  
 {  
 p=p->aft;  
 }  
 }  
 else  
 {  
 p=tail;  
 for(int i=0;i<size-index;i++)  
 {  
 p=p->pre;  
 }  
 }  
 return p->val;  
 }  
   
 void addAtHead(int val) {  
 addAtIndex(0,val);  
 }  
   
 void addAtTail(int val) {  
 addAtIndex(size,val);  
 }  
   
 void addAtIndex(int index, int val) {  
 if(index<-1||index>size) return;  
 dll \*p,\*q;//p代表前，q代表后  
 if(index<size-index)//也就是在head离他更近一点  
 {  
 p=head;  
 for(int i=0;i<index;i++)  
 {  
 p=p->aft;  
 }  
 q=p->aft;  
 }  
 else  
 {  
 q=tail;  
 for(int i=0;i<size-index;i++)  
 {  
 q=q->pre;  
 }  
 p=q->pre;  
 }  
 size++;  
 dll \*toadd=new dll(val);  
 toadd->pre=p;  
 toadd->aft=q;  
 p->aft=toadd;  
 q->pre=toadd;  
 }  
   
 void deleteAtIndex(int index) {  
 if(index<0||index>=size) return;  
 dll \*p,\*q;//p代表前，q代表后  
 if(index<size-index)//也就是在head离他更近一点  
 {  
 p=head;  
 for(int i=0;i<index;i++)  
 {  
 p=p->aft;  
 }  
 q=p->aft->aft;  
 }  
 else  
 {  
 q=tail;  
 for(int i=0;i<size-index-1;i++)  
 {  
 q=q->pre;  
 }  
 p=q->pre->pre;  
 }  
 size--;  
 dll\* tod=p->aft;  
 p->aft=q;  
 q->pre=p;  
 delete tod;  
  
 }  
 private:  
 int size;  
 dll \*head;  
 dll \*tail;  
};  
/\*\*  
 \* Your MyLinkedList object will be instantiated and called as such:  
 \* MyLinkedList\* obj = new MyLinkedList();  
 \* int param\_1 = obj->get(index);  
 \* obj->addAtHead(val);  
 \* obj->addAtTail(val);  
 \* obj->addAtIndex(index,val);  
 \* obj->deleteAtIndex(index);  
 \*/

结果：



